# Temporal Prologの一実現について

2011/JUN/20

たけおか
@takeoka

# 時相論理

# よくある時制の様相記号

a⇒○b　ある時刻にaが成り立つと、次の時刻にbが成り立つ

a⇒□b　ある時刻にaが成り立つと、その後ずっとbが成り立つ

a⇒◇b　ある時刻にaが成り立つと、次の後いつかはbが成り立つ

a⇒ b until c　ある時刻にaが成り立つと、次の後cが成り立つまでは、bが成り立つ

# Temporal Prolog

- Temporal Prologは、桜川貴司氏が考案した時相論理型言語である。

http://ci.nii.ac.jp/naid/110003743453

- 時間は離散的
  1. 一つの時刻に、リダクションすべき事柄をすべて行い、
  2. そこで真になった事実を過去データベースにアサートし、
  3. 時刻を進める
  4. 1に戻る

# Temporal Prologの時制の様相記号 C(condition)

●c　前の時刻にcが成り立った(前の時刻が無い場合はfalse)

■c　現在までずっとcが成り立っている

◆c　現在までにcが成り立ったことがある

c since d　最も最近にdが成り立って以来、その時刻も含めて現在までcは常に成り立っている

c after d　最も最近にdが成り立って以後、その時刻も含めて現在までにcが成り立ったことがある

c for n　現在も含めて過去n回連続してcが成り立った

# Temporal Prologの時制の様相記号 R(result)

r until a　aが成り立つ直前の時刻までrが成り立つ

r atnext a　aが初めて成り立つ時、r も成り立つ

□r　未来永劫r が成り立つ

# Temporal Prologの各時相記号の●への変換

a⇒○b

○b :- a.

ある時刻にaが成り立つと、次の時刻にbが成り立つ

b :- ●a.

一つ前の時刻にaが成り立つと、現在、bが成り立つ

# Temporal Prologの各時相記号の●への変換

●●●●a⇒b

b :- ●●●●a.

は、

b :- ●a3.

a3 :- ●a2.

a2 :- ●a1.

a1 :- ●a.

# Temporal Prologの各時相記号の●への変換

q :- ●●●●p,print(“asd”).

((q)(●p3 *x) (print "asd"))

((p3 *y) (●p2 *x))

((p2 *x) (●p1 *x))

((p1 *x) (●p *x))

((p *x) (at 3))

((gomi) (●q))))

# Temporal Prologの各時相記号の●への変換

```
*ima-mukashi*=NIL *at_0*=NIL
*yaraneba*=(((●Q) (●P3 *X) (PRINT "asd")) ((●P *X) (AT 3)) ((●P1 *X) (●P *X))
        ((●P2 *X) (●P1 *X)) ((●P3 *Y) (●P2 *X)))
**** 1-shikiri **** *time*=0
*hito-mukashi*=NIL
*yaraneba*=(((●Q) (●P3 *X) (PRINT "asd")) ((●P *X) (AT 3)) ((●P1 *X) (●P *X))
        ((●P2 *X) (●P1 *X)) ((●P3 *Y) (●P2 *X)))
**** 1-shikiri **** *time*=1
*hito-mukashi*=NIL
**** 1-shikiri **** *time*=2
*hito-mukashi*=NIL
**** 1-shikiri **** *time*=3
*hito-mukashi*=NIL
**** 1-shikiri **** *time*=4
*hito-mukashi*=(((●P *X)))
**** 1-shikiri **** *time*=5
*hito-mukashi*=(((●P1 *X)))
**** 1-shikiri **** *time*=6
*hito-mukashi*=(((●P2 *X)))
**** 1-shikiri **** *time*=7
*hito-mukashi*=(((●P3 *Y)))
"asd"
**** 1-shikiri **** *time*=8
*hito-mukashi*=(((●Q)))
**** 1-shikiri **** *time*=9
*hito-mukashi*=NIL
**** 1-shikiri **** *time*=10
*hito-mukashi*=NIL
```

# Temporal Prologの各時相記号の●への変換

□b

b :-.

ずっとbが成り立つ

a⇒□b

□b :- a.

ある時刻にaが成り立つと、その後ずっとbが成り立つ

p:-a.

p:- ●p.

b :- p.

# Temporal Prologの各時相記号の●への変換

a⇒◇b

ある時刻にaが成り立つと、次の後いつかはbが成り立つ

◇b :- c,a.

b:- c, p.

p :- ●p.　　% pは、過去に一回でも成立した事を表す

p :- a.

# Temporal Prologの各時相記号の●への変換

q until a

ある時刻にaが成り立つと、次の後qが成り立つまでは、bが成り立つ

q :- ~a.

a⇒ q until c

ある時刻にaが成り立つと、次の後cが成り立つまでは、qが成り立つ

p:-a.

p:-●p,●~c.　%aが過去に成立し、前回cは成立していない

q:-p,~c.

# Temporal Prologの各時相記号の●への変換

q atnext b

bが初めて成り立つ時、qも成り立つ

q:-b.

a⇒ q atnext b

ある時刻にaが成り立つと、bが初めて成り立つ時、qも成り立つ

p:-a.

p:-●p, ●~b. %aが過去に成立し、前回bは成立していない

q:- p,b.

# よくある
# 時相論理言語の
# 実装

# 時相論理言語のよくある実装(誤解).1

- 任意の時刻の過去を参照する

  ⇒過去に成立した事実を記憶する

  ⇒探索空間がリニアに成長

  ⇒記憶領域と探索時間が、いつかは爆発

  ⇒プログラミング言語としては、実用性無し(という烙印)

# 時相論理言語のよくある実装.2

- すべての節をリダクションする

  ⇒明らかに無駄な節をリダクション

  ⇒実用性無し

# たけおかのTemporal Prolog実装

# 記憶について

- 任意の時刻の過去を参照する
- すべての過去を覚える(のか?)

が、しかし、

- Tempral Prologでは、すべての時相が一つ前(●)に変換できる

  と、桜川が述べている
- 一つ前の過去しか記憶しなくて良い
  - 二つ前 (●●p == p2)は、

    p2 :- ●p1.

    p1 :- ●p.

    となる。

    節2つとして記憶される。元来必要な記憶のみで、無駄は無い
- 結論

<u>一つ前(*ひと昔*)のみを記憶する</u>

# 記憶について

## 一つ前の過去しか記憶しなくて良い

- 二つ前 (●●p == p2)は、

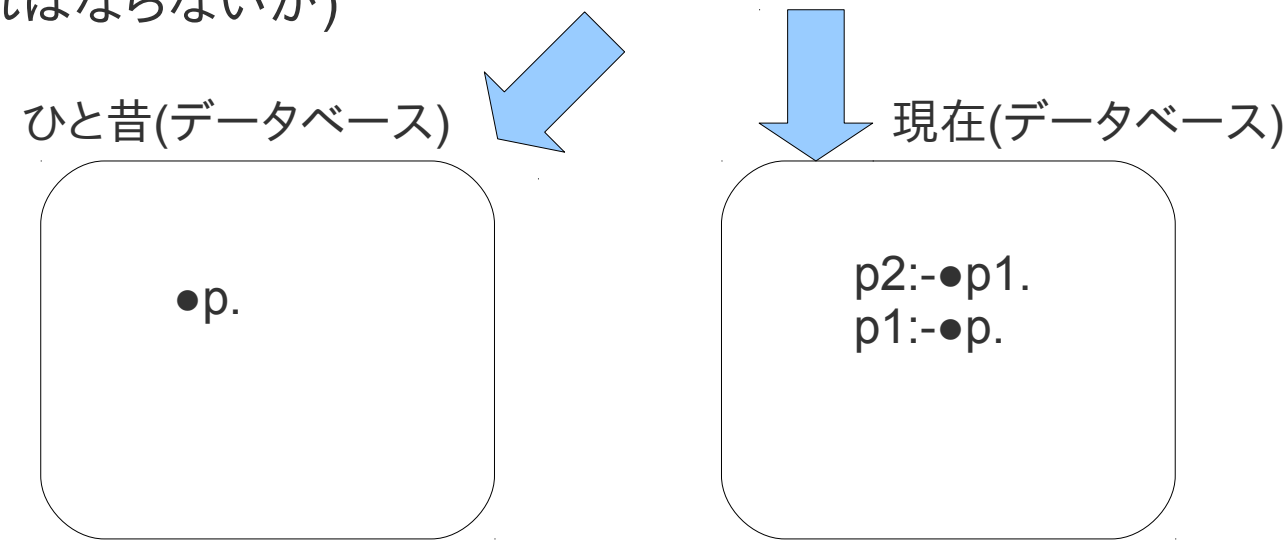
  p2 :- ●p1.

  p1 :- ●p.

  となる。

  節2つとして記憶される。元来必要な記憶のみで、無駄は無い

処理系は、ゴール節中の項に●があれば、*ひと昔*を探索。通常は、現在のDBを探索。
(実は、●を付けて、現在のデータベースにassertしても良い。毎時刻、ふた昔前をretractしなければならないが)



# 記憶について

- 結論

  一つ前(*ひと昔*)のみを記憶する

- 探索空間は通常のPrologの倍
  - 成長しない
  - 非常に小さい
  - 通常のPrologは、assertやretractがあるが、
    Temporal Prologにはそういうものが無いので、おあいこ(?)
  - *ひと昔*は、毎時刻クリア&アサートされる

# 駆動について

- 毎時刻、すべての節をリダクションするのか?

- 要求駆動としたい
  - 理論的な言語の処理系の実現は、

    大抵、要求駆動になっている
  - 俺もそういうかっこいい言語処理系を作りたいと思ってたんだよ(笑)

# 駆動について

- 要求駆動としたい
- そもそもTemporal Prologは何をするひとなのか?
  - 一つ前の過去が参照されるから、それが必要なだけでは?

```
p(Y) :- ●p(X), append(X,[0],Y).
append([], Y, Y).
append([A|X], Y, [A|Z]) :- append(X,Y,Z).
plus1(X,Z) :- Z is X + 1 .
```

    - ここでappend/3やplus1(X)が、でたらめな引数でリダクションされても、誰も嬉しくない
  - 次の時刻に参照されるべき節だけをリダクションすればよい

# 駆動について

- 要求駆動としたい
- そもそもTemporal Prologは何をするひとなのか?
  - 一つ前の過去が参照されるから、それが必要なだけでは?
    - 仕様記述の面からは、それは違う
    - だが、実行を実用的に行う言語としてはアリ
      - 無駄が無くて素晴らしい

# 駆動について

- 要求駆動としたい
- 一つ前の過去が参照されるから、それが必要なだけ
- 次の時刻に参照されるべき節だけをリダクションすればよい
- ボディ部中で、●が付いているファンクタの節だけをリダクション

```
p(Y) :- ●p(X), append(X,[0],Y).
append([], Y, Y).
append([A|X], Y, [A|Z]) :- append(X,Y,Z).
plus1(X,Z) :- Z is X + 1 .
```

# 駆動について

- 次の時刻に参照されるべき節だけをリダクションすればよい
  - ボディ部中で、●が付いているファンクタの節だけをリダクション
  - ●で参照されることが無い節は、実行されない(リダクション)
  - ただのprint()を実行させたければ、●で依存関係を明示的に書く
    - まぁ、それぐらいはいいよね

      qq :- print('asd').

      pp :- ●qq.
  - 本来なら、ppが何かから参照されなければ、●qqも不要なので、実行されないべきであろう
    - しかし、それでは、ここでのprintf/1が実行されず、悲しい
  - ∴ボディ部中で●が付いた節は、無条件にリダクション
    - 依存関係を厳密に見るなら、そもそも大元で最初の要求を起こすものが必要

# おしまい

- http://www.takeoka.org/~take/

http://www.takeoka.org/~take/ailabo/prolog/tmpro/tmpro-memo.html

http://www.takeoka.org/~take/ailabo/prolog/tmpro/tmpro-man.html

- プログラム

http://www.takeoka.org/~take/ailabo/prolog/tmpro.tgz

- 桜川のTemporal Prologについての文章

http://ci.nii.ac.jp/naid/110003743453